

DC Line Modeling in Transient Stability

DC Line models for time-domain simulations come in two broad categories. The first category does not model the firing angle dynamics and assumes that once a choice of a DC voltage and DC current are made the DC line converters will instantaneously change the firing angles to the values that would achieve a desired DC voltage and DC current schedule. These models also do not include any of the DC-side line dynamics created by a potential RLC circuit of the DC transmission system. Examples of dynamic models the use this treatment are CDC1T, CDC4T, CDC6, CDC6T, CEEL2T, CHATGY, CMDWS2T, EPCDC, and RSPDC3.

Another type of model will include the converter firing angle dynamics and the DC side transmission line dynamics. Examples of this are the CHVDC2 two-terminal model and the special multi-terminal DC line models MTDC_IPP and MTDC_PDCI along with their MTDC Converter models.

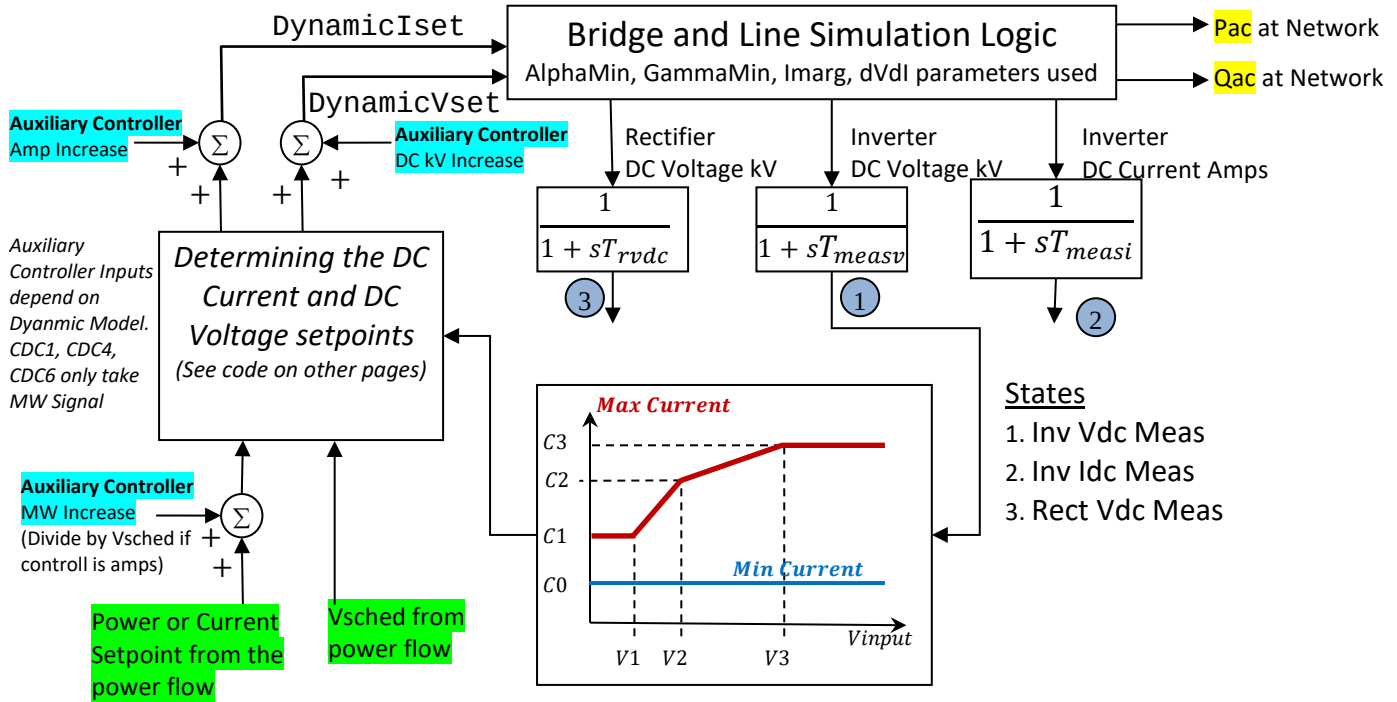
This document covers how PowerWorld Simulator implements these types of models in the transient stability numerical simulation. This will help you better understand how to configure the input parameters.

Table of Contents

1	Algebraic Treatment of Converters with Control Dynamics.....	2
1.1	Bridge and Line Simulation Algebraic Logic.....	3
1.2	Logic for Determining the DC Current and DC Voltage Setpoint	5
1.3	Initialization of Variables at the beginning of the simulation.....	5
1.4	Code run at in the algebraic portion of each time step solution.....	6
2	Detailed Model with Including DC firing Angle Dynamics and DC Line Side Dynamics.....	11
2.1	DC Converter Equations Revisited.....	11
2.2	Implementation of the numerical solution of the PDCI.....	12
2.3	Numerical Integration to model the change in DC currents and DC voltages after a change in the $\cos\alpha$ and $\cos\beta$ terms ..	14
2.4	Solution of the algebraic change in DC voltages during the AC network boundary equation solution.....	16

1 Algebraic Treatment of Converters with Control Dynamics

The overall control structure of DC lines is something like the following depending on the model. The implementation of the Voltage-Dependent Current Order Limit can be different as some use measurements of DC voltage and others use AC side per unit voltages. Some models permit Auxiliary Controller inputs. Most models will only take a MW increase input and do not provide input to the DC current or DC voltage.

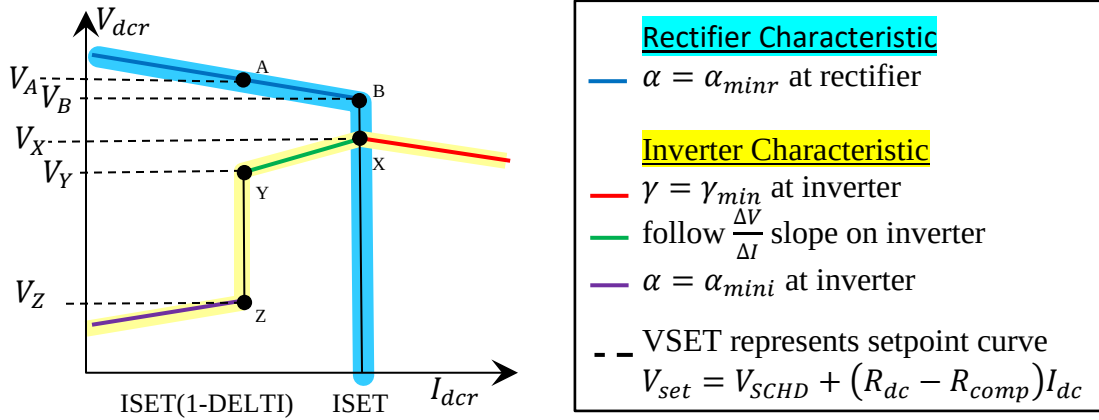


The input parameters for models such as this include the following parameters

Bridge and Line Simulation Logic	AlphaMin, GammaMin (degrees) DELTI is the Current Margin (comes from the power flow record normally) dVdI (0.0 unless dynamic model includes parameter)
Measurement delays	Tmeasv, Tmeasi, Tvrdc (seconds)
Parameters for Voltage-Dependent Current Order Limiter (VDCOL)	C0, C1, C2, C3 (parameter in Amps) V1, V2, V3 (parameters in kV)
Block and Unblock rectifier without a time delay on under AC rectifier voltage	Vblock (per unit AC voltage for blocking) Vunbl (per unit AC voltage for unblocking)
How long a rectifier will remain blocked once decision is made to block it.	Tblock (seconds)
Bypass and Unbypass inverter without a time delay on under DC inverter voltage	Vbypas (per unit DC voltage for bypass)
How long inverter will remain bypassed once a decision is made to bypass it	Tbypass (seconds)
Parameters use when restarting a DC line after performing a block or bypass action	Rsvolt (DC kV) Rscur (DC Amps) Vramp (pu/second) (per unit of DynamicVset) Cramp (pu/second) (per unit of DynamicIset)
Block and unblock using an AC voltage <i>with a time delay in the decision</i>	Vdeblk (per unit AC voltage) Trblck (time delay for blocking) Treblk (time delay for unblocking use Vunbl)
Block Rectifier based on the inverter AC voltage including a communication time delay	Vinblk (per unit AC voltage) Tinblk (time delay for block decision in seconds) Tcomb (communication time delay)
Bypass and Unpass inverter	Vacbyp (per unit AC voltage) Tdebyp (time delay for bypass decision in seconds)

1.1 Bridge and Line Simulation Algebraic Logic

The operating point of the DC line is somewhere on the **Rectifier Characteristic** but is limited by the **Inverter Characteristic**. Also the DC voltage setpoint curve (Vset below) will interact with these curves to get the operating point too.



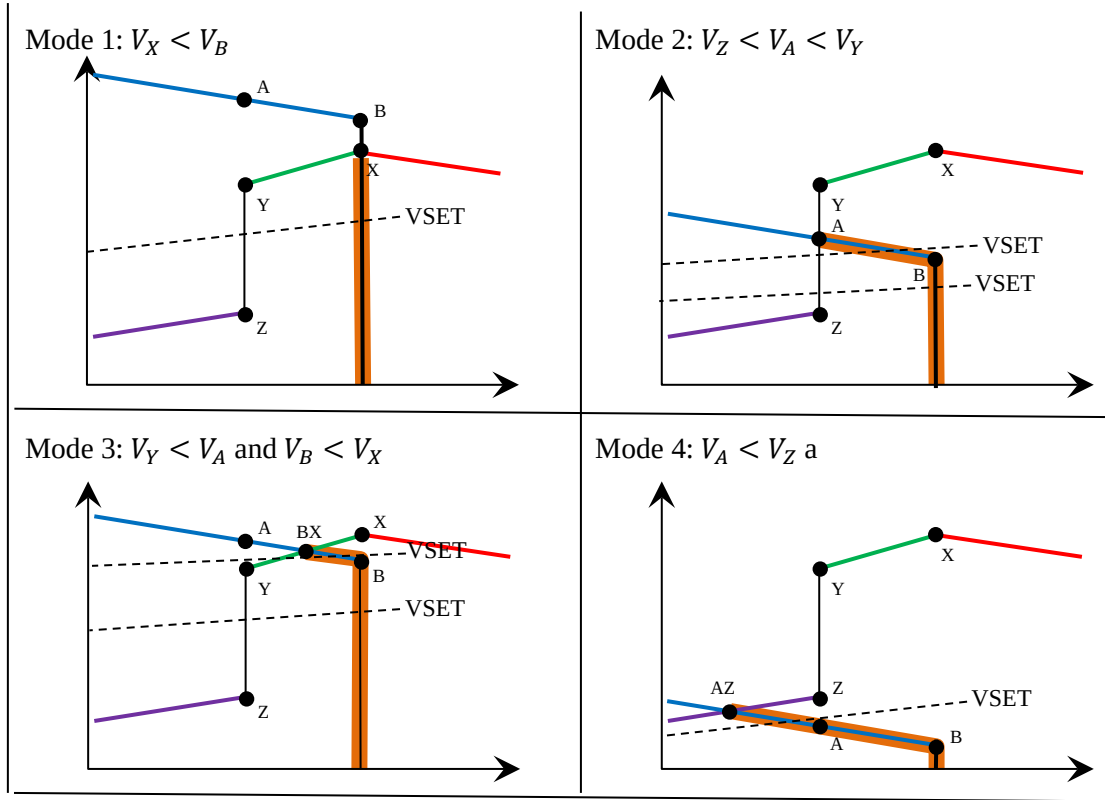
The characteristics of the DC line and converters (DC resistance, angle limits, tap ratios, AC terminal voltage, etc...) determine the lines for the inverter and rectifier characteristics operating an minimum firing angles. Along with those input, additional control inputs as follows come from the model.

- VSET: Voltage setpoint for the DC Line
- ISET : Current setpoint for the DC Line.
- DELTI : Delta I which represents the current margin
- dVdI : slope between VY and VX. For most DC line models this is 0.0, but for some dynamic models this is an input parameter (for example CHIGATT, CEEL2T, and CMDWS2T)

To find the operating point, we calculate the DC voltages at the 5 points A, B, Y, X, and Z. These 5 points represents points of intersection between various curves as follows.

- V_A is the voltage on the minimum rectifier firing angle (α_{minr}) where current is $ISET*(1-DELTI)$
- V_B is the voltage on the minimum rectifier firing angle (α_{minr}) where current is ISET
- V_X is the voltage on the minimum inverter firing angle (γ_{min}) where the current is to ISET
- V_Y is the voltage that slopes away from VX using the slope dVdI where the current is $ISET*(1-DELTI)$
- V_Z is the voltage on the minimum inverter alpha curve (α_{mini}) where current is $ISET*(1-DELTI)$. As of 2021, all dynamic models that use this calculation assume a $\alpha_{mini} = 90 \text{ degrees}$. This comes into play when faulting the terminal of a DC line
- V_{BX} is the intersection between the line from A to B and the line from Y to X
- V_{AZ} is the intersection of the minimum inverter alpha curve and the minimum rectifier alpha curve

Using these points, we can then progress through logic to get the appropriate DC Voltage and DC Current. We will be operating somewhere on the **orange** shaded regions. If the Voltage setpoint such that $V_{SET} < V_B$ or $V_X < V_B$ then we are on the vertical portion where $I_{dcr} = I_{SET}$. If the operating point is not on this vertical segment, then we are operating wherever the blue α_{minr} curve intersects the inverter characteristic. This intersection point depends on the mode 2, 3, or 4 as depicted in the following figure.



Put into pseudo-code we can write the following generally.

```

If (VSETR <= VB) or (VX <= VB) then begin
  // This handles Mode 1 and all other mode situations sitting on ISET
  Idc := ISET;
  Vdc := VX;
  If Vdc > VSETR then Vdc := VSETR;
End
Else begin
  if VA > VY then Vdc := Vax // MODE 3
  Else if VA < VZ then Vdc := Vaz // MODE 4
  Else Vdc := VA // MODE 2
  If Vdc > VSETR then Vdc := VSETR;
  Idc := ISET*(1-DeltI*(Vdc-VB)/(VA-VB))
End;

```

1.2 Logic for Determining the DC Current and DC Voltage Setpoint

```
// the following is a global variable
PRESENT_Time          : Time          // This is the present simulation time

// The following are internal algebraic variables stored with the Dynamic Model
fDynamicISet          : Float         // this is the DC Current setpoint sent to the bridge logic
fDynamicVSet          : Float         // this is the DC Voltage setpoint sent to the bridge logic
fRectifierBlocked     : boolean       // Set to TRUE when the rectifier is blocked
fInverterBypassed     : boolean       // Set to TRUE when the inverter is bypassed

// The following are internal algebraic variables used for determining whether to
// BLOCK or UNBLOCK the rectifier terminal
fPerformingRestartFromBlock : boolean
fTimeOfRestartBlock        : Time
fTimeOfBlock               : Time
fTimeBelowVdeblk           : Time
fTimeBelowVdeblkEnabled    : boolean
fTimeBelowVinblk           : Time
fTimeBelowVinblkEnabled    : boolean
fTimeAboveVunby            : Time
fTimeAboveVunbyEnabled     : boolean
fUnBlockReason             : Options are (dccc6BlockInstant, dccc6BlockDelay, dccc6BlockInverter)

// The following are internal algebraic variables used for determining whether to
// BYPASS or UNBYPASS the inverter terminal
fPerformingRestartFromBypass : boolean
fTimeOfRestartBypass         : Time
fTimeOfBypass                : Time
fTimeBelowVacbyp             : Time
fTimeBelowVacbypEnabled      : boolean
fUnBypassReason              : Options are (dccc6BRBypassInstant, dccc6BRBypassDelay);

// The following are internal algebraic variables used to force the switch from power to current control
fForceCurrentMode           : boolean
fTimeOfForceCurrentMode     : TPWTimeStamp
```

1.3 Initialization of Variables at the beginning of the simulation

```
fPerformingRestartFromBlock := false;
fTimeOfRestartBlock         := 0;
fTimeOfBlock                := 0;
fTimeBelowVdeblk            := 0;
fTimeBelowVdeblkEnabled     := false;
fTimeBelowVinblk            := 0;
fTimeBelowVinblkEnabled     := false;
fTimeAboveVunby             := 0;
fTimeAboveVunbyEnabled      := false;
fUnBlockReason              := dccc6BlockInstant;

fPerformingRestartFromBypass := false;
fTimeOfRestartBypass         := 0;
fTimeOfBypass               := 0;
fTimeBelowVacbyp            := 0;
fTimeBelowVacbypEnabled     := false;
fUnBypassReason              := dccc6BRBypassInstant;

fTimeOfForceCurrentMode     := 0;
fForceCurrentMode           := false;
```

1.4 Code run at in the algebraic portion of each time step solution

```
Get from DC Line the Control Settings From the steady state power flow record
(
  local_PowerSpecifiedAtRectifier : boolean // TRUE if power setpoint is at Rectifier
  local_PISetpoint                : Float   // Current Schedule
  local_VSCHED                    : Float   // Voltage Schedule
  local_ControlMode               : CURRENT_MODE or POWER_MODE
  local_VCMODE                   : Float    // DC voltage at which POWER_MODE is abandoned for CURRENT_MODE
)

//-----
// This section will depend on the dynamic model. Most Models do this except CEEL2T and CMDWS2T
//-----
if "Dynamic Model takes a MW power input from the Auxiliary Controller" then begin
  theAuxMW := Get Signal From Auxiliary Controller Signal Index 1;
  if local_ControlMode = CURRENT_MODE
  then local_PISetpoint := local_PISetpoint + theAuxMW/local_VSCHED
  else local_PISetpoint := local_PISetpoint + theAuxMW; // everything is in MW
end;

//-----
// Figure out the desired VSet and ISet from the algebraic model of the DC Line
//-----
fDynamicVSet := local_VSCHED;
if (local_ControlMode = POWER_MODE) and (not fForceCurrentMode) then begin
  if STATE_InvVdcMeas < local_VCMODE then begin
    fForceCurrentMode := true;
    fTimeOfForceCurrentMode := PRESENT_Time;
  end;
end
else if fForceCurrentMode then begin
  // Wait for at least Tcmode seconds, but after that as soon as VinVMeas exceeds VCMODE
  // switch back to constant power mode
  if (STATE_InvVdcMeas >= local_VCMODE) and ( (PRESENT_Time - fTimeOfForceCurrentMode) > fTcmode ) then begin
    fForceCurrentMode := false;
  end;
end;
If local_ControlMode = CURRENT_MODE Then begin
  fDynamicISet := local_PISetPoint;
end
Else Begin // Otherwise constant power - get current from the specified value and actual voltage
  if fForceCurrentMode then begin
    // force current mode, so use value based on VCMODE
    fDynamicISet := local_PISetPoint/local_VCMODE;
  end
  else begin
    If local_PowerSpecifiedAtRectifier
    Then local_V := STATE_RectVdcMeas
    Else local_V := STATE_InvVdcMeas;
    fDynamicISet := local_PISetPoint/local_V;
  end;
end;
End;

// Following logic is needed to handle setting the input of the dynamic Current Setpoint in models
// that have a time delay in the block diagram for the current setpoint
// This includes the CEEL2T and CMDW2ST models
SetDynamicISetStateInput(fDynamicISet); // Needed if the model uses a Delay Block
fDynamicISet := GetDynamicISetState;    // This may be the fDynamicISet directly, or after a delay block

if "Dynamic Model takes a Current input from the Auxiliary Controller" then begin
  fDynamicISet := fDynamicISet + Get Signal From Auxiliary Controller Signal Index 1;
end;

if "Dynamic Model takes a Voltage input from the Auxiliary Controller" then begin
  fDynamicVSet := fDynamicVSet + Get Signal From Auxiliary Controller Signal Index 2;
end;
```

```

//-----
// Handle Decisions regarding Blocking the DC Line Rectifier
//-----
local_RectACVolt := RectACVoltMag; // Get the present AC Voltage at rectifier AC bus
local_InvACVolt  := InvACVoltMag;  // Get the present AC Voltage at inverter AC bus
local_InvDCVolt  := InvDCVolt;     // Get the present DC Voltage at inverter

LOCAL_JustUnblockedIt := false;
if fRectifierBlocked then begin
  case fUnBlockReason of
    dccdc6BlockInstant : if (local_RectACVolt > fVunbl) then begin
                          if ( (PRESENT_Time - fTimeOfBlock) > fTblock ) then begin
                            fPerformingRestartFromBlock := true;
                            fTimeOfRestartBlock := PRESENT_Time;
                            fRectifierBlocked := false;
                            LOCAL_JustUnblockedIt := true;
                          end;
                        end;
    dccdc6BlockDelay    : if (local_RectACVolt > fVunbl) then begin
                          if ( (PRESENT_Time - fTimeOfBlock) > fTreblk ) then begin
                            fPerformingRestartFromBlock := true;
                            fTimeOfRestartBlock := PRESENT_Time;
                            fRectifierBlocked := false;
                            LOCAL_JustUnblockedIt := true;
                          end;
                        end
                        else begin
                          fTimeOfBlock := PRESENT_Time; // push time ahead again
                        end;
    dccdc6BlockInverter : begin
                          // just waiting until it goes above Vunby.
                          // When it does start the timer
                          if local_InvACVolt > fVunBy then begin
                            if not fTimeAboveVunbyEnabled then begin
                              fTimeAboveVunby := PRESENT_Time;
                              fTimeAboveVunbyEnabled := true;
                            end;
                          end;
                          // If the timer is going, then check to see if it's expired.
                          if fTimeAboveVunbyEnabled
                            and ( (PRESENT_Time - fTimeAboveVunby) > fTcomb )
                          then begin
                            fPerformingRestartFromBlock := true;
                            fTimeOfRestartBlock := PRESENT_Time;
                            fRectifierBlocked := false;
                            fTimeAboveVunbyEnabled := false;
                            LOCAL_JustUnblockedIt := true;
                          end;
                        end;
  end;
end
else if not fPerformingRestartFromBlock then begin
  //-----
  // fVBlock = rectifier ac blocking voltage INSTANTANEOUS
  if local_RectACVolt < fVBlock then begin
    fRectifierBlocked := true;
    fTimeOfBlock := PRESENT_Time;
    if FlipUnBlockReasonForRectifier
    then fUnBlockReason := dccdc6BlockDelay
    else fUnBlockReason := dccdc6BlockInstant;
    if fTimeBelowVinblkEnabled then fTimeBelowVinblkEnabled := false;
    if fTimeBelowVdeblkEnabled then fTimeBelowVdeblkEnabled := false;
  end;
  //-----
  // fVdeblk = DELAYED rectifier ac blocking
  if (not fRectifierBlocked) and (local_RectACVolt < fVdeblk) then begin
    if not fTimeBelowVdeblkEnabled then begin
      fTimeBelowVdeblk := PRESENT_Time; // just keep pushing this forward in time!
      fTimeBelowVdeblkEnabled := true;
    end;
    if ( (PRESENT_Time - fTimeBelowVdeblk) > fTdeblk ) then begin
      fRectifierBlocked := true;
      fTimeOfBlock := PRESENT_Time;
      if FlipUnBlockReasonForRectifier
      then fUnBlockReason := dccdc6BlockInstant
      else fUnBlockReason := dccdc6BlockDelay;
      fTimeBelowVdeblkEnabled := false;
      if fTimeBelowVinblkEnabled then fTimeBelowVinblkEnabled := false;
    end;
  end;
end;

```

```

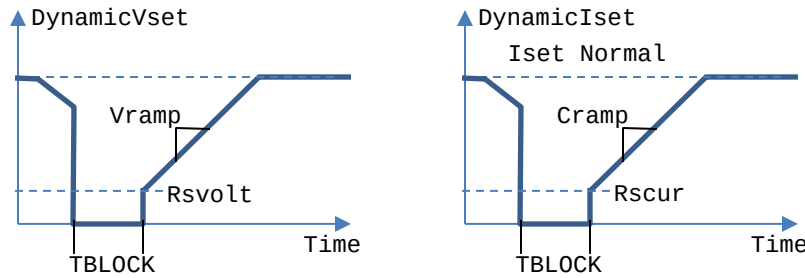
    end;
end
else begin
    fTimeBelowVdeblk := PRESENT_Time; // just keep pushing this forward in time!
    fTimeBelowVdeblkEnabled := false; // disable timer
end;
//-----
// fVinblk - block passed on inverter ac voltage. Decision is instant, but delay due to communication
if (not fRectifierBlocked) then begin
    if (local_InvACVolt < fVinblk) then begin
        if not fTimeBelowVinblkEnabled then begin
            fTimeBelowVinblk := PRESENT_Time;
            fTimeBelowVinblkEnabled := true;
        end;
    end;
    // Once time is enabled it can NOT be stopped.
    // We're just waiting for the timer to expire here
    // fTcomb here is just representing the communication delay
    if fTimeBelowVinblkEnabled and ( (PRESENT_Time - fTimeBelowVinblk) > fTcomb ) then begin
        fRectifierBlocked := true;
        fTimeOfBlock := PRESENT_Time;
        fUnBlockReason := dccdc6BlockInverter;
        fTimeBelowVinblkEnabled := false;
        if fTimeBelowVdeblkEnabled then fTimeBelowVdeblkEnabled := false;
    end;
end
else begin
    fTimeBelowVinblk := PRESENT_Time;
    fTimeBelowVinblkEnabled := false;
end;
end;
end;

```

```

//-----
// Handle Decisions regarding Bypassing the DC Line Inverter
//-----
if fRectifierBlocked or LOCAL_JustUnblockedIt then begin
    // don't try to bypass while it's already blocked!
end
else begin
    if fInverterBypassed then begin
        case fUnBypassReason of
            dccdc6BRBypassInstant : if (local_InvACVolt > fVunby) and
                                     ( (PRESENT_Time - fTimeOfBypass) > fTbypas )
                                     then begin
                                         fPerformingRestartFromBypass := true;
                                         fTimeOfRestartBypass := PRESENT_Time;
                                         fInverterBypassed := false;
                                     end;
            dccdc6BRBypassDelay : if (local_InvACVolt > fVunby) then begin
                                     if ( (PRESENT_Time - fTimeOfBypass) > fTinbyp ) then begin
                                         fPerformingRestartFromBypass := true;
                                         fTimeOfRestartBypass := PRESENT_Time;
                                         fInverterBypassed := false;
                                     end;
                                     end
                                     else begin
                                         fTimeOfBypass := PRESENT_Time;
                                     end;
        end;
    end
    else if not fPerformingRestartFromBypass then begin
        //-----
        // fVbypas = INSTANT inverter DC voltage bypass
        if (local_InvDCVolt < fVbypas) then begin
            fInverterBypassed := true;
            fTimeOfBypass := PRESENT_Time;
            fUnBypassReason := dccdc6BRBypassInstant;
        end;
        //-----
        // fVacbyp = DELAYED inverter ac bypass
        if (not fInverterBypassed) and (local_InvACVolt < fVacbyp) then begin
            if not fTimeBelowVacbypEnabled then begin
                fTimeBelowVacbyp := PRESENT_Time; // just keep pushing this forward in time!
                fTimeBelowVacbypEnabled := true;
            end;
            if ( (PRESENT_Time - fTimeBelowVacbyp) > fTdebyp ) then begin
                fInverterBypassed := true;
                fTimeOfBypass := PRESENT_Time;
                fUnBypassReason := dccdc6BRBypassDelay;
            end;
        end
        else begin
            fTimeBelowVacbyp := PRESENT_Time; // just keep pushing this forward in time!
            fTimeBelowVacbypEnabled := false; // disable timer
        end;
    end;
end;
end;

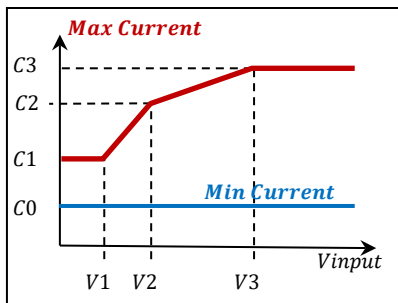
```



```

//-----
// Modify VSet or ISet when restarting from block
//-----
if fPerformingRestartFromBlock then begin
    local_RampIset := fRscur + (PRESENT_Time - fTimeOfRestartBlock) * fCramp*fDynamicIset; // fCramp in pu/sec
    if (local_RampIset < fDynamicIset)
        and (fCramp > 0) // just ignore ramp and jump immediately if fCramp <= 0
    then begin
        fDynamicIset := local_RampIset; // bring down to "ramp programming"
        local_IRecovered := false;
    end
    else begin
        local_IRecovered := true;
    end;
    local_RampVset := fRsvolt + (PRESENT_Time - fTimeOfRestartBlock) * fVramp*fDynamicVset; // fVramp in pu/sec
    if (local_RampVset < fDynamicVset)
        and (fVramp > 0) // just ignore ramp and jump immediately if fVramp <= 0
    then begin
        fDynamicVset := local_RampVset; // bring down to "ramp programming"
    end
    else begin
        if local_IRecovered then fPerformingRestartFromBlock := false; // we're fully recovered!
    end;
end
else begin
    if fDynamicIset < fC0 then fDynamicIset := fC0;
end;
//-----
// Modify VSet when restarting from bypass. Current reset not needed for bypass
//-----
if fPerformingRestartFromBypass then begin
    local_RampVset := fRsvolt + (PRESENT_Time - fTimeOfRestartBypass) * fVramp*fDynamicVset; // fVramp in pu/sec
    if (local_RampVset < fDynamicVset)
        and (fVramp > 0) // just ignore ramp and jump immediately if fVramp <= 0
    then begin
        fDynamicVset := local_RampVset; // bring down to "ramp programming"
    end
    else begin
        fPerformingRestartFromBypass := false; // we're fully recovered!
    end;
end;
end;

```



```

//-----
// Put in Voltage-Dependent Current Order Limiter (VDCOL)
// do lookup max current using V1, C1, V2, C2, V3, C3
//-----
local_V := STATE_InvVdcMeas;
if local_V > STATE_RectVMeas then local_V := STATE_RectVMeas
if (local_V <= fv1) then local_MaxLimit:= fC1
else if (local_V <= fv2) and (fv2 > fv1) then local_MaxLimit:= fC1 + (local_V-fv1)/(fv2-fv1)*(fC2-fC1)
else if (local_V <= fv3) and (fv3 > fv2) then local_MaxLimit:= fC2 + (local_V-fv2)/(fv3-fv2)*(fC3-fC2)
else local_MaxLimit:= fC3;
if fDynamicIset > local_MaxLimit then fDynamicIset := local_MaxLimit;

```

2 Detailed Model with Including DC firing Angle Dynamics and DC Line Side Dynamics

2.1 DC Converter Equations Revisited

Normally the rectifier and inverter equations are written in terms of the *firing angles* α and γ .

Rectifier Equations	Inverter Equations
$V_{dcr} = \frac{3\sqrt{2}N}{\pi} \left[\frac{D_{base}}{Tap} \right] V_{ac} \cos(\alpha)$ $- N \left[\frac{3X_c}{\pi} + 2R_c \right] I_{dc} - N[V_{diode}]$ $E_{ac} = \left[\frac{D_{base}}{Tap} \right] V_{ac}$ $\mu = \cos^{-1} \left[\cos \alpha - \frac{\sqrt{2}I_{dc}X_c}{E_{ac}} \right] - \alpha$	$V_{dci} = \frac{3\sqrt{2}N}{\pi} \left[\frac{D_{base}}{Tap} \right] V_{ac} \cos(\gamma)$ $+ N \left[-\frac{3X_c}{\pi} + 2R_c \right] I_{dc} + N[V_{diode}]$ $E_{ac} = \left[\frac{D_{base}}{Tap} \right] V_{ac}$ $\mu = \cos^{-1} \left[\cos \gamma - \frac{\sqrt{2}I_{dc}X_c}{E_{ac}} \right] - \gamma$

For transient stability purposes it is more convenient to write the inverter equations in terms of β . β is related to γ by $\beta = \mu + \gamma$. During the simulation we will need to flip back and forth between β and γ , thus express β as a function of γ and vice versa.

$$\beta = \cos^{-1} \left[\cos \gamma - \frac{\sqrt{2}I_{dc}X_c}{E_{ac}} \right] \quad \text{and} \quad \gamma = \cos^{-1} \left[\cos \beta + \frac{\sqrt{2}I_{dc}X_c}{E_{ac}} \right].$$

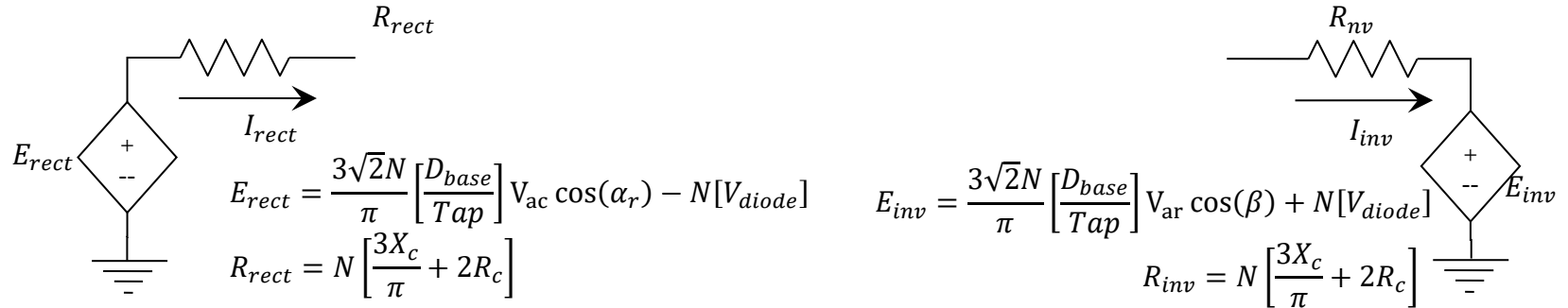
Now using the relationship $\cos \gamma = \cos \beta + \frac{\sqrt{2}I_{dc}X_c}{E_{ac}}$, rewrite the DC voltage equation for the inverter in terms of β .

$$V_{dci} = \frac{3\sqrt{2}N}{\pi} E_{ac} \left[\cos \beta + \frac{\sqrt{2}I_{dc}X_c}{E_{ac}} \right] + N \left[-\frac{3X_c}{\pi} + 2R_c \right] I_{dc} + N[V_{diode}]$$

$$V_{dci} = \frac{3\sqrt{2}N}{\pi} E_{ac} \cos \beta + N \left[\frac{6X_c}{\pi} I_{dc} \right] + N \left[-\frac{3X_c}{\pi} + 2R_c \right] I_{dc} + N[V_{diode}]$$

$$V_{dci} = \frac{3\sqrt{2}N}{\pi} E_{ac} \cos \beta + N \left[\frac{3X_c}{\pi} + 2R_c \right] I_{dc} + N[V_{diode}]$$

Using these equations, model the converter in the DC network equations as follows, with the extra constraint that the currents can NOT be negative.



2.2 Implementation of the numerical solution of the PDCI

In the normal network boundary equations and in Simulator's power flow engine, the DC converter control is assumed to be instantaneous. We assume that firing angle move instantaneously to bring DC currents instantaneously to a new operating point. For the PDCI we will be removing these assumptions completely and modeling the dynamics of the firing angle control. It is convenient for numerical reasons to make the control output of the DC converter equal to either $\cos(\alpha)$ and $\cos(\beta)$ depending on whether the converter is acting as a rectifier or inverter. Here we will describe how this changes the numerical simulation of the multi-terminal DC line.

Inside the transient stability engine in Simulator, an explicit integration method is used. Thus the general process of solving the equations is to use a numerical integration time step to update all dynamic state variables and then another step to update all the algebraic variables such as the AC system voltage and angle (network boundary equations). The MTDC simulation is added into this framework as follows.

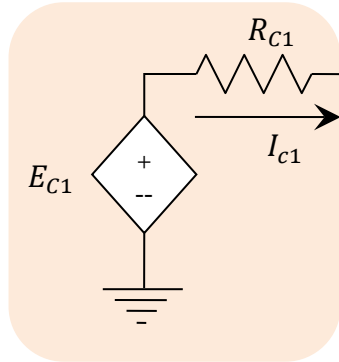
During each time-step of the numerical simulation of the multi-terminal DC Line, the following steps are taken

1. Using the standard routines within Simulator to calculate the new states for the dynamic models MTDC_PDCI, CONV_CELILO_E, CONV_CELILO_N, and CONV_SYLMAR. This is done along with all the other thousands of dynamic models.
--> Updated Variables are $\cos(\alpha)$ or $\cos(\beta)$ terms for the DC converters
2. Before solving all the other algebraic equations using the AC network boundary equations, take the new $\cos(\alpha)$ and $\cos(\beta)$ terms and use them to model a step change in the DC voltages seen by the DC network equations. Use numerical integration to simulate the change in DC voltages and DC currents on the transmission lines for this time-step. Note that this is the only place in the numerical routines where the DC system currents in the system change.
--> Updated Variables are the DC voltages and DC Currents

Finally, when the normal AC network boundary equations are being solved we must modify how the DC network equations are handled. Without the dynamic model for the MTDC line modeled, we assume that the DC currents respond instantaneously to a change in the DC voltages by an instantaneous change in the converter firing angles. Because the PDCI is modeling the actual dynamics of the DC converter firing angle change, we must assume that the firing angles remain constant during the network boundary equation solution instead. In addition, the PTDC is a very long transmission line and thus has a substantial inductance, therefore the current cannot change instantaneously either and must be assumed constant during the network boundary equation solution. This means that in order to solve the DC network equations we must allow the voltages in the DC network to change but since we're not allowing the currents to change, then this means all voltage changes from resistances must be zero. This means that the new voltage variations must be coming from th

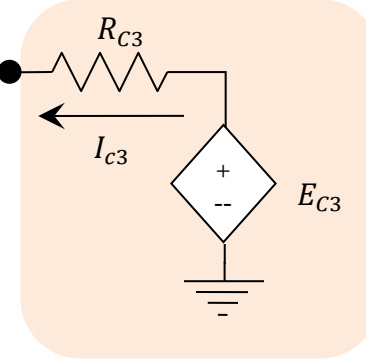
Multi-Terminal DC Network DC Network Equations

$$R_{C1} = N_{C1} \left[\frac{3X_{cC1}}{\pi} + 2R_{cC1} \right]$$



$$E_{C1} = \frac{3\sqrt{2}N}{\pi} \left[\frac{D_{base}}{Tap} \right] V_{acC1} \cos(\alpha_{C1}) - N[V_{diode}]$$

$$R_{C3} = N_{C3} \left[\frac{3X_{cC3}}{\pi} + 2R_{cC3} \right]$$



$$E_{C3} = \frac{3\sqrt{2}N}{\pi} \left[\frac{D_{base}}{Tap} \right] V_{acC3} \cos(\alpha_{C3}) - N[V_{diode}]$$

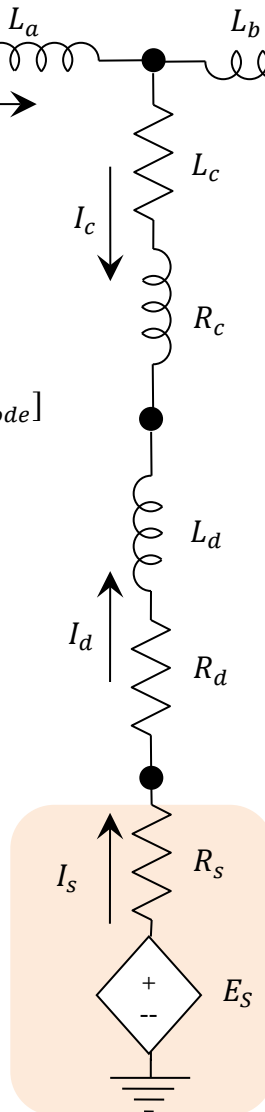
Following are AC per unit voltages from the network equations.

$$V_{acS}, V_{acC3}, V_{acC1}$$

Following are outputs of the various dynamic DC converter models.

$$\cos(\beta_S), \cos(\alpha_{C1}), \cos(\alpha_{C1})$$

Note: For coding purposes it is easier to flip the direction of the current at the inverter and then just require that it be negative instead of positive

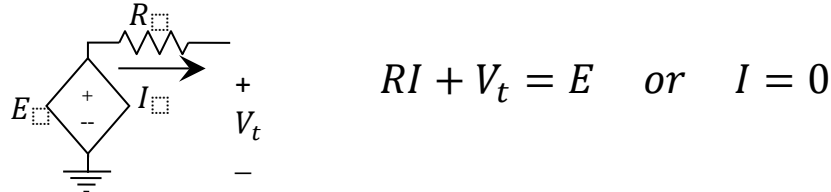


$$R_S = N_S \left[\frac{3X_{cS}}{\pi} + 2R_{cS} \right]$$

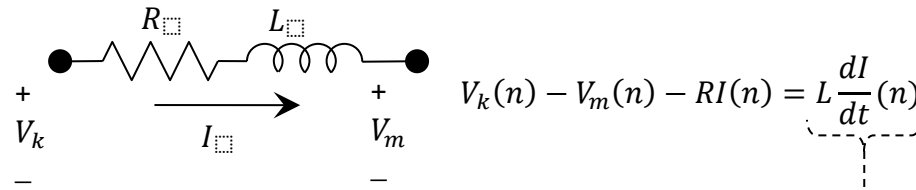
$$E_S = \frac{3\sqrt{2}N}{\pi} \left[\frac{DC_{base}}{Tap} \right] V_{acs} \cos(\beta_S) + N[V_{diode}]$$

2.3 Numerical Integration to model the change in DC currents and DC voltages after a change in the $\cos(\alpha)$ and $\cos(\beta)$ terms

DC Converter equations are as follows, where R and E were described earlier. The unknown variable associated with this equation is the DC current. Note that it's possible for this to result in a current that is impossible (negative current injected into DC network for a rectifier or positive injection for an inverter). In that case we use a different equation which sets the current to zero.



DC transmission line equations are as follows. The unknown variable associate with this equation is the DC line current.



Use the trapezoidal rule to write the $dI/dt(n)$ term

$$V_k(n) - V_m(n) - RI(n) = \frac{2L}{h}I(n) - \left[\frac{2L}{h}I(n-1) + L \frac{dI}{dt}(n-1) \right]$$

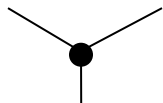
Use the standard relationship above to write the $LdI/dt(n-1)$ term

$$V_k(n) - V_m(n) - RI(n) = \frac{2L}{h}I(n) - \left[\frac{2L}{h}I(n-1) \right] - [V_k(n-1) - V_m(n-1) - RI(n-1)]$$

Finally, group the terms that are a function of values at time (n) on the left and time (n-1) on the right

$$V_k(n) - V_m(n) + \left(-R - \frac{2L}{h} \right) I(n) = \left(+R - \frac{2L}{h} \right) I(n-1) - V_k(n-1) + V_m(n-1)$$

DC Bus equation is just Kirchhoff's Current Law (KCL). The unknown variable associated with the equation is the DC bus voltage.



$$I_a + I_b + I_c + \dots = 0$$

Remember the trapezoidal integration rule:

$$\frac{\frac{dx}{dt}(n) + \frac{dx}{dt}(n-1)}{2} = \frac{x(n) - x(n-1)}{h}$$

which gives

$$\frac{dx}{dt}(n) = \frac{2}{h}x(n) - \left[\frac{2}{h}x(n-1) + \frac{dx}{dt}(n-1) \right]$$

Where n = the present integer time step

h = the duration of the time step

$x(n)$ = value at present time

$x(n-1)$ = value at previous time step

Using the PDCI as an example, the following matrix equations are created.

	Ic1	Ic3	Is	Ic	Ia	Ib	Id	v3	v4	v7	v8	v9	X	B
Celilo1	Rc1									1			Ic1 (n)	Ec1
Celilo3		Rc3									1		Ic3 (n)	Ec2
Sylmar1			Rs									1	Is (n)	Es
LineC				$(-Rc-2Lc/h)$				1	-1				Ic (n)	$(+Rc-2Lc/h)*Ic_{(n-1)}$ $- v3_{(n-1)} + v4_{(n-1)}$
LineA					$(-Ra-2La/h)$			-1		1			Ia (n)	$(+Ra-2La/h)*Ia_{(n-1)}$ $- v7_{(n-1)} + v3_{(n-1)}$
LineB						$(-Rb-2Lb/h)$		-1			1		Ib (n)	$(+Rb-2Lb/h)*Ib_{(n-1)}$ $- v8_{(n-1)} + v3_{(n-1)}$
LineD							$(-Rd-2Ld/h)$		-1			1	Id (n)	$(+Rd-2Ld/h)*Id_{(n-1)}$ $- v9_{(n-1)} + v4_{(n-1)}$
KCL3				-1	1	1							v3 (n)	0
KCL4				1			1						v4 (n)	0
KCL7	1				-1								v7 (n)	0
KCL8		1				-1							v8 (n)	0
KCL9			1				-1						v9 (n)	0

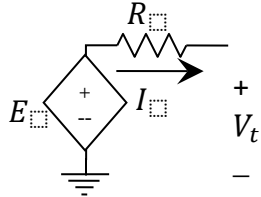
Simulator solves this set of equations by splitting the actual integration time-step used in the standard numerical integration and dividing it by 10. We set the h variable above to the time step divided by 10 then iterate this set of equations 10 times. When solving these equations, we initially assume that none of the currents end up the wrong sign. Then after each sub time step we check if the converter currents end up as the wrong sign. If converter currents have the wrong sign, we automatically change the equation for the offending converter to force that converter current to zero and redo this sub time step. We assume the converter current remains zero during the remainder of this time step and only allow it to back-off this limit during the following time step. I believe the current would never bounce around during a time step anyway, because this is fundamentally only a set of RL circuits so you will only get a first-order RL circuit exponential decay response toward the new steady state.

As an example, if the current at Celilo1 ended up the wrong sign, then its matrix equation would be rewritten as

Celilo1	1									0			Ic1 (n)	0
---------	---	--	--	--	--	--	--	--	--	---	--	--	---------	---

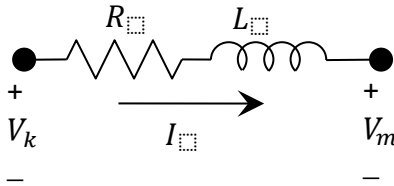
2.4 Solution of the algebraic change in DC voltages during the AC network boundary equation solution

DC Converter equations are as follows, where R and E were described earlier. Also note that the DC current is assumed constant so it is moved to the right side as a known quantity. The unknown variable associated with this equation is the voltage at the terminal.



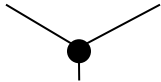
$$V_t = E - RI$$

DC transmission line equations are as follows. The unknown variable associated with equation is the *derivative* of the DC current. The DC current is assumed constant during the AC network boundary equation solution.



$$V_k - V_m - L \frac{dI}{dt} = RI$$

DC Bus equation is just Kirchhoff's Current Law (KCL), but for the derivatives of the currents instead of the actual currents. The unknown variable associated with the equation is DC bus voltage. Also note that we only add an equation for DC buses which are not connected to a DC converter terminal.



$$\frac{dI_a}{dt} + \frac{dI_b}{dt} + \frac{dI_c}{dt} + \dots = 0$$

Using the PDCI as an example, the following matrix equations are created

LineC	-Lc						1	-1	=	dIc/dt	Ic*Rc
LineA		-La			1		-1			dIa/dt	Ia*Ra
LineB			-Lb			1	-1			dIb/dt	Ib*Rb
LineD				-Ld			1	-1		dId/dt	Id*Rd
Celilo1					1					v7	Ec1-Rc1*Ic1
Celilo3						1				v8	Ec3-Rc3*Ic3
Sylmar1							1			v9	Es-Rs*Is
KCL3	-1	1	1					0		v3	0
KCL4	1			1				0		v4	0